KILPATRICK TOWNSEND & STOCKTON LLP
JAMES G. GILLILAND, JR. (State Bar No. 107988)
jgilliland@kilpatricktownsend.com
MEHRNAZ BOROUMAND SMITH (State Bar No. 197271)
mboroumand@kilpatricktownsend.com
STEVEN D. MOORE (State Bar No. 290875)
smoore@kilpatricktownsend.com
ROBERT J. ARTUZ (State Bar No. 227789)
rjartuz@kilpatricktownsend.com
BENJAMIN M. KLEINMAN-GREEN (State Bar No. 261846)
bkleinman-green@kilpatricktownsend.com
KEVIN J. O'BRIEN (State Bar No. 278823)
kobrien@kilpatricktownsend.com
Eighth Floor, Two Embarcadero Center
San Francisco, CA 94111
Telephone: 415 576 0200
Facsimile: 415 576 0300

KILPATRICK TOWNSEND & STOCKTON LLP
BRYAN STUART FOSTER (*pro hac vice*)
bfoster@kilpatricktownsend.com
1001 West Fourth Street
Winston-Salem, NC 27101-2400
Telephone: 336-607-7300
Facsimile: 336-607-7500

Attorneys for Defendants
ORACLE CORPORATION, ORACLE AMERICA, INC.
and ORACLE INTERNATIONAL CORPORATION

UNITED STATES DISTRICT COURT

FOR THE NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

THOUGHT, INC., a California corporation, Plaintiff, v. ORACLE CORPORATION, a Delaware corporation; ORACLE AMERICA, INC., a Delaware corporation; and ORACLE INTERNATIONAL CORPORATION, a California corporation, Defendants.	Civil Action No. C12-5601 WHO **ORACLE'S MEMORANDUM OF POINTS AND AUTHORITIES IN SUPPORT OF ITS MOTION FOR SUMMARY JUDGMENT** Date: April 20, 2016 Time: 2:00 p.m. Ctrm.: 2, 17th Floor Hon. William H. Orrick

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

Page

TABLE OF AUTHORITIES

TABLE OF AUTHORITIES
(continued)

I. INTRODUCTION

Plaintiff Thought, Inc.(“Thought”) originally accused Defendants (collectively, “Oracle”) of infringing seven patents relating to software technology called object relational mapping (“ORM”). Six of those patents have since been dismissed because they were either found invalid in *inter partes* review proceedings or abandoned by Thought after Oracle showed in discovery that they are not infringed and are invalid. Only U.S. Patent No. 5,857,197 (the “’197 patent”) remains, as to which Thought alleges that Oracle infringes claims 1, 3, 5, 7 and 8 (the “Asserted Claims”). As demonstrated below, the Court should relegate this patent to the same fate as the other six.

The accused versions of Oracle’s TopLink product do not infringe the ’197 patent because they use a different software architecture than the claimed invention. The claimed invention requires a specific split-interface architecture to perform ORM, where separate and distinct interfaces (or adapters) are responsible for performing particular functions including, for example, “extracting” information from a software object, “packing” and “unpacking” data for communication between interfaces or adapters, and “instantiating” new objects. The accused versions of TopLink do not use a split-interface architecture and do not perform many of the functions that the Asserted Claims require. Consequently, the Court should grant summary judgment of non-infringement.

While the accused versions of TopLink do not infringe, a prior art version designed over 20 years ago operated the same way as the claimed invention and renders the Asserted Claims invalid. In 1994, TopLink’s original developer, The Object People, released the first version of TopLink[1], written in a programming language called Smalltalk. TopLink for Smalltalk 2.0 (“TLS2”) was released, sold, and used in the United States in 1996, well before the March 1997 invention date of the ’197 patent. TLS2 anticipates all of the Asserted Claims or, at a minimum, renders them obvious. Therefore, the Court should also grant summary judgment of invalidity.

[1] Unless indicated otherwise, the term “TopLink” refers to the accused software versions that were written in Java and sold beginning in 2006 (which include both TopLink and a related open-source software application known as “EclipseLink”). The prior art version of TopLink that invalidates the ’197 patent will be referred to as TopLink for Smalltalk 2.0 or TLS2.

II. STATEMENT OF FACTS

A. The ’197 Patent and Object Relational Mapping Technology

ORM software has existed since at least the early 1990s to facilitate the transfer of data between object-oriented software applications (“object applications”) and relational databases. (Declaration of Antony L. Hosking, Ph.D, in Support of Oracle’s Motion for Summary Judgment (“Hosking Decl.”), ¶ 26.) An object application is a software program (e.g., a web browser) that is written in an object-oriented programming language (e.g., C++, Java, or Smalltalk), and that works with data in the form of objects.[2] A relational database, in contrast, is software that organizes and stores data using data tables organized in rows and columns. Object data and relational data are not directly compatible. ORM software (a type of “middleware”) sits between object applications and relational databases and translates object data to a relational format and vice versa. (*Id.*)

The basic operation of the claimed invention is identical to that of preexisting ORM systems. (*Id.* at ¶ 29.) For example, to fulfill an application’s request to read data from a relational database both the claimed invention and prior art systems work as follows: (1) a user defines the rules (referred to as “metadata”) that map an application’s object classes to a relational database’s tables, (2) the application invokes the ORM system to request data, (3) the ORM system uses the metadata to translate the application’s request into a database-compatible form, (4) the ORM system accesses the database, (5) the ORM system uses the metadata to convert the retrieved database content into an object, and (6) the ORM system sends the object to the application. (*Id.*)

Thought does not dispute that ORM technology was well known and used before March 1997 to perform the above functions. Rather, Thought asserts that the ’197 patent claims a novel architecture to perform ORM. (*Id.* at ¶¶ 32-33.) Specifically, the claimed invention requires a software architecture in which specific ORM functions are performed by two separate and distinct components called (depending on the Asserted Claim in question) either “interfaces” or “adapters.”[3]

---

[2] An object is an instance of a class, and a class is a category of things that have similar properties and can be acted on in identical ways. For example, all instances of an Employee class have names and titles, and all can be hired and fired. These instances, or objects, represent specific employees.

[3] Claims 1 and 7 recite “interfaces.” Claims 3 and 8 recite “adapters.” Claim 5, which depends from both claims 1 and 3, incorporates both alternatives. Under the Court’s constructions (in which all

(*Id.*) Each of these components has a very specific role in handling a request received from an object application, and each request must be accompanied by an object that stores information about the requested relational data. (Declaration of Robert J. Artuz in Support of Oracle's Motion for Summary Judgment ("Artuz Decl."), Ex. 1 ['197 patent] at 35:7-11.) The first interface (or adapter) processes the object – including extracting information from the object and packing it as data – and communicates the request and packed data to a second, separate, interface or adapter. (*Id.* at 35:5-15 and 36:01-07.) This second interface (adapter) is responsible for performing translation and accessing the database as described in steps (3) and (4) above. (*Id.* at 35:15-22 and 36:07-16.) The second interface (adapter) must also pack the results from the database (*id.* at 36:19-21) and send them to the first interface (adapter), which then: (1) unpacks those results and converts them into an object (a step the claims call instantiating an object) and (2) communicates the object back to the object application. (*Id.* at 35:14-18 and 36:21-28.)

While this split-interface architecture was Thought's claimed invention, it is not necessary for ORM and is not used in TopLink. (Hosking Decl., ¶¶ 45-49.) Rather, the relevant components of TopLink are indivisible and do not pack or perform other claimed functions, because they do not need to do so. (*Id.* at ¶¶ 50-51.)

Under Thought's theories, when the accused TopLink functions are invoked, various TopLink components respond. While Thought labors to divide these components into separate first and second interfaces (or adapters), they are not. There is no logical division between these components – they are connected, run in the same logical space, and can directly invoke each other's functions. (*Id.*; Dkt. 76-1 [Jagadish's Claim Construction Reply Brief] at ¶ 14; Declaration of Dennis Leung in Support of Oracle's Motion for Summary Judgment ("Leung Decl."), ¶ 29.) Moreover, they do not perform all of the claimed functions. (Hosking Decl., ¶¶ 51-58.)

B. Thought Accuses Oracle Products Sold with a License to TopLink

Thought's infringement allegations focus on Oracle's sales of TopLink. (Artuz Decl., Ex. 2 [Feb. 8, 2016 deposition of Hosagrahar Jagadish] at 161:3-19; Ex. 13 [Opening Report of

claims require adapters) and the parties' contentions, there is no material difference between an adapter and an interface for the purposes of Oracle's non-infringement arguments.

Hosagrahar Jagadish] at ¶¶ 616-20 and 626-27.) Oracle has rarely sold TopLink as a standalone software product. (Leung Decl., ¶ 24.) Rather, a TopLink license is typically included with Oracle's licenses to other products that fall within the "middleware" category. (*Id.* at ¶¶ 24 and 26.) Oracle middleware includes a variety of software including products called Internet Application Server and WebLogic, which can be used to build and host object applications.[4] (*Id.* at ¶¶ 26 and 27.)

Thought alleges that Oracle's sale of *any* product that includes a license to TopLink[5] infringes the '197 patent. (Artuz Decl., Ex. 5 at 42:17-44:10 and 68:24-70:9.) However, TopLink might not be used for ORM. (Leung Decl., ¶¶ 25-27.) Some of its features support ORM while others support non-ORM functions such as object-to-XML mapping and object caching. (*Id.*; *see also* Hosking Decl., ¶ 60.) Moreover, TopLink can perform ORM functions in different ways, not all of which are accused of infringement. (*Id.*) Finally, irrespective of Thought's accusations, TopLink uses neither the split-interface architecture of the claimed invention nor certain functions that the claims require. (Hosking Decl., ¶¶ 50-58.)

C. Early Versions of TopLink Are Prior Art to the '197 Patent

One of the pioneers of ORM technology was The Object People, which was founded in 1991 by three Carleton University professors in Ottawa, Canada. The Object People delivered ORM solutions for external clients before creating TopLink in 1994. (Leung Decl., ¶¶ 2 and 4.) Early versions of TopLink were written in an object-oriented programming language called Smalltalk. (*Id.* at ¶ 3.) TLS2 was released in early 1996 and sold to and used by customers in Canada and the United States. (*Id.* at ¶¶ 5-12, 15, and 17-20.)

Unlike the accused Java versions of TopLink, TLS2 operated identically to the claimed invention of the '197 patent. Although TLS2 was publicly known and used in the United States well before the '197 patent's March 20, 1997 filing (*id.*), it was not considered by the Patent Office when the '197 patent was examined for patentability. (Hosking Decl., ¶ 100.) Thought does not dispute that TLS2 practiced all the elements of the Asserted Claims except one. (Hosking Decl., ¶¶ 66-73.) Thought contests only whether TLS2's adapters were "runtime interchangeable" as required

[4] A full list of the accused products in this case can be found at Artuz Decl., Ex. 7.
[5] Thought specifically accuses Oracle middleware products that are sold with a license to TopLink versions 10-12. (Artuz Decl., ¶ 44.)

by the Court's claim construction. (Dkt. 116 at 10:22-12:21.) Thought's argument, however, is based on a clear misunderstanding of the undisputed facts concerning the design and operation of TLS2, and cannot prevent summary judgment.

In 1996, The Object People began developing a version of TopLink for Java, which at the time was a new object-oriented programming language.[6] (*Id.* at ¶¶ 21-22.) TopLink for Java 1.0 was released in 1997. (*Id.* at ¶ 22.) In 2000, WebGain, Inc. acquired the TopLink business and continued to release new versions.[7] (*Id.* at ¶ 23.) In 2002, Oracle acquired the TopLink portion of WebGain's business and also developed and released new versions. (*Id.*) The versions of TopLink that Thought accuses of infringement – versions 10-12 – are all written in Java. (*Id.*)

III. TOPLINK DOES NOT INFRINGE THE ASSERTED CLAIMS

A. Legal Standard for Summary Judgment of Non-infringement

Summary judgment is appropriate when no genuine issue of material fact exists, and the moving party is entitled to judgment as a matter of law. Fed. R. Civ. P. 56(a); *Celotex Corp. v. Catrett,* 106 S. Ct. 2548, 2552 (1986). Thought alleges only literal infringement, which requires that the accused system meets each and every claim limitation. *Playtex Prods., Inc. v. Procter & Gamble Co*., 400 F.3d 901, 906 (Fed. Cir. 2005). To survive a motion for summary judgment of non-infringement, a patentee must show that "features of the accused product would support a finding of infringement under the claim construction adopted by the court[.]" *Intellectual Sci. and Tech. Inc. v. Sony Elecs., Inc.*, 589 F.3d 1179, 1183 (Fed. Cir. 2009). However, "[c]onclusory expert assertions cannot raise triable issues of material fact on summary judgment." *Sitrick v. Dreamworks, LLC,* 516 F.3d 993, 1001 (Fed. Cir. 2008).

B. Because TopLink Has a Different Architecture from the Asserted Claims, the Accused Products Do Not Infringe

The Asserted Claims require a specific architecture, and TopLink performs ORM using a different one. Thought tries to make a case for infringement by forcing labels (e.g., "adapters" and

[6] Java was first announced in 1995 and released in 1996. It was promoted as an object-oriented programming language that could be used by a wide range of developers. Java's release and rapid adoption encouraged many developers, including The Object People, to create products for the Java development market. (Leung Decl., ¶ 21.)

[7] TLS was discontinued by WebGain after the release of TLS 5.1. (Leung Decl., ¶ 23.)

"interfaces") from the Asserted Claims onto components of TopLink, but those components simply do not match up with the claimed elements and do not perform all of the claimed functions.

Specifically, and as shown below, TopLink does not infringe because: (1) what Thought alleges to be the claimed first interface or adapter does not perform the functions that the Asserted Claims require of that component; (2) TopLink does not "pack" data as required by the Asserted Claims; and (3) TopLink does not "extract" the required object name and attributes. Moreover, Oracle's sales of TopLink cannot give rise to a claim of direct infringement by Thought.

1. The Asserted Claims Require a Specific Split-Interface Architecture

The Asserted Claims require two distinct interfaces or adapters, each of which must perform separate tasks. The first interface or adapter extracts information from an object received from the object application and packs the extracted information as data for communication to the second interface/adapter. (*See, e.g.*, Artuz Decl., Ex. 1 at 35:10-15 and 36:1-6.) The second interface/adapter then processes the data, along with metadata, to create and execute a command that retrieves the requested content from the database. (*Id.* at 35:16-21 and 36:5-17.) The second interface/adapter then packs the database content as data and communicates it to the first interface/adapter (*id.* at 36:19-21), which then unpacks the data and instantiates (creates) a new object for delivery to the object application. (*Id.* at 35:13-16 and 36:23-28.)

The division between the first and second interfaces/adapters is shown as item 650 in Figure 1 of the '197 patent and is described as a "logical division." (*Id.* at Fig. 1 and 7:51-54.) One reason for this logical division is that the two interfaces/adapters may need to exist on separate computers or networks. (*See, e.g.*, *id.* at 4:15-21, 4:29-32, 4:41-48, 4:49-5:12, 9:34-44; Hosking Decl., ¶¶ 33-36.) The logical division also explains the need to pack and unpack data before and after it is communicated between the interfaces/adapters. (Hosking Decl., ¶¶ 37-39.) Although having the first and second interfaces/adapters on separate computers or networks is not required by the Court's claim constructions, the claimed packing and unpacking makes possible an implementation of the claimed invention that has such a physical division. (*Id.*)

2. TopLink Does Not have Distinct First and Second Interfaces/Adapters that Perform the Claimed Functions

TopLink does not include the split-interface architecture of the claimed invention. In a typical TopLink deployment – including all of the scenarios accused of infringement – all relevant TopLink components run on the same computer or network and in the same process. There is no logical or physical division among them. (Leung Decl., ¶ 29.) Thought cannot show that one set of TopLink components performs all of the functions of the first interface or adapter and that a separate set of TopLink components performs all of the functions of the second interface or adapter.

(a) TopLink does not have a first interface or adapter that both extracts and instantiates

As noted above, the first interface/adapter must: (1) extract an object name and object attributes from an object (Artuz Decl., Ex. 1 at 35:10-15 and 36:1-6) and (2) instantiate (create) a new object based on content retrieved from the database. (*Id.* at 35:13-16 and 36:23-28.) The TopLink components Thought alleges are the first interface/adapter do not perform both tasks.

For example, Thought identifies a TopLink component called EntityManager as the first claimed interface/adapter that allegedly performs the requisite extraction and instantiation functions. (Hosking Decl., ¶ 57.) While Oracle disputes that EntityManager (or any other component Thought points to as the first interface/adapter[8]) performs the extraction (*see* Section III.B.4 *infra*), Thought's infringement analysis fails because neither TopLink's EntityManager nor any other alleged first interface/adapter ***also instantiates*** a new object based on the retrieved database content. (*Id.*) Instead, for the instantiation step Thought points to a different TopLink component, a "DatabaseAccessor". (*Id.*) But Thought does not, and cannot, contend that this component is the first interface/adapter. (*Id.*) In fact, the DatabaseAccessor passes the instantiated object back to the EntityManager (and the other components alleged to be the first interface/adapter). (*Id.*) Because the alleged first interface/adapter does not perform the claimed instantiation step, TopLink does not infringe.

[8] Thought's expert contends some components other than EntityManager also constitute the first interface/adapter, but those allegations fail for exactly the same reason. (Hosking Decl., ¶¶ 55-58.)

(b) TopLink does not have a first interface/adapter that packs and unpacks

Two other functions of the first interface/adapter are: (1) packing the object name and attributes as data (so they can be sent across the logical division to the second interface/adapter) (Artuz Decl., Ex. 1 at 35:10-15 and 36:1-6) and (2) unpacking the retrieved database content (sent to it in packed form by the second interface/adapter). (*Id.* at 35:13-16 and 36:23-28; Hosking Decl., ¶ 58.)

The TopLink components that Thought alleges to be the first interface/adapter do not perform both of these tasks.[9] Thought identifies the unpacking step as the performance of TopLink's "valueFromRow" method, which is an entirely different component from what Thought contends to be the first interface/adapter (e.g., the EntityManager). This valueFromRow method, according to the analysis performed by Thought's expert, is never invoked by anything Thought labels as the alleged first interface/adapter. (Hosking Decl., ¶ 58.) Accordingly, the alleged first interface/adapter does not perform the required unpacking step.

3. Because TopLink's Architecture is Not Split, Packing and Unpacking is Unnecessary and Not Performed

Even if Thought could show that TopLink had the claimed interfaces or adapters, it cannot show packing or unpacking. The Court has construed "packing" to mean "placing data into a form suitable for communication between software components" and "unpacking" to mean "retrieving data from a packed data structure." (Dkt. 116 at 8:6-16.) Packing data may be beneficial and efficient for communication of data under circumstances where software components are split across a logical division, such as the separate computers or networks disclosed in the '197 patent. (Hosking Decl., ¶¶ 37-39.) But packing is not necessary if a system simply wants to communicate data among components running on the same computer and in the same software process. (*Id.* at ¶¶ 50-51.) Because TopLink does not use the split-interface architecture of the claimed invention, it does not use the claimed packing (and unpacking) functions. Rather, when TopLink's components communicate with each other they simply send data as is, without first "placing data into a form

[9] Also, none of the alleged first interfaces/adapters that Thought identifies performs the claimed packing function (*see* Section III.B.3 *infra*).

suitable for communication," or packing the data.

Thought alleges that TopLink packs information as data when the value of one object is set to have the value of another object. (Hosking Decl., ¶ 54c.) But this is not packing because the allegedly "packed" data does not change from the form it had before it was allegedly packed. (*Id.*) The Court's construction requires a process by which the data is manipulated in some fashion so that it is placed into a "form suitable for communication." (Dkt. 116 at 6:6-7 (For packing to occur, "data does not necessarily need to be 'translated' for communication. It could, instead, be compacted, reordered, or otherwise manipulated for various purposes.").) The named inventor of the '197 patent, Ward Mullins, acknowledged as much. When trying to sell his patents to non-practicing entities, he said that "packing" means processing the object into more "primitive data types." (Artuz Decl., Ex. 6 at THOR00086541.)

Packing does not merely send along data in its current form; some change must occur to place it into a form suitable for communication. If the data in question remains in the same form before and after it is communicated, as is the case with TopLink, no packing has occurred. (Hosking Decl., ¶ 54c.) To find otherwise would improperly render the packing and unpacking steps of the Asserted Claims superfluous. *See MicroStrategy Inc. v. Bus. Objects, S.A.*, 429 F.3d 1344, 1351-52 (Fed. Cir. 2005). Thus, TopLink does not infringe for this reason as well.

4. TopLink Does Not Infringe Because it Does Not Practice the Claimed Step of "Extracting … from the Object"

The Asserted Claims also require "extracting the object attributes and the object name from the object" that the object application sends to the ORM system as part of its request. (Artuz Decl., Ex. 1 at 35:64-36:2.) However, (1) Thought's alleged first interface or first adapter does not "extract . . . object attributes" from an object, but merely passes the entire object; and (2) TopLink does not obtain the object attributes and object name from "the object", but rather obtains the first from one object and the second from a second object.

First, the object attributes are not "extracted" by Thought's alleged first interface or adapter at all. Instead, what Thought alleges is the first interface or adapter in TopLink simply passes an entire object received from the object application (which allegedly contains the object attributes) without extracting anything from that object. (Hosking Decl., ¶ 54a.) Just as one extracts coal from a mine, but not a mine from a mine, merely passing an entire object does not extract object attributes from that object. (*Id.*)

Second, in its only infringement theory, Thought relies on TopLink's operation as an implementation of the Java Persistence API ("JPA") standard.[10] (*Id.* ¶ 54b.) According to this theory, "an object comprising object attributes and an object name" is received by TopLink when its retrieval functionality is invoked using a JPA method called "find." (*Id.*) However, as Thought acknowledges and as required by the JPA standard, when TopLink is invoked using the "find" method, TopLink actually receives two objects. (*Id.*) Thought also admits that the purported extraction of an "object name" is from one of these objects, and the alleged extraction of the "object attributes" is from the other. (*Id.*) In other words, although the claims require the object name and object attributes be extracted from "the object" (i.e., the single object sent by the object application to the first interface or adapter), Thought concedes that they are extracted (if at all) from separate objects. TopLink, therefore, does not meet the literal language of the claims, which require the object name and object attributes to come from the same, single object – "the object" sent to the ORM system.

Thought asserts, without any explanation or factual basis, that the two objects are collectively one object. (*See, e.g.*, Artuz Decl., Ex. 14 at p. 55.) Because they are demonstrably two distinct objects, Thought's unfounded, conclusory opinion cannot prevent summary judgment.

C. Because TopLink Does Not Infringe, No Accused Products Infringe

Although Thought accuses approximately 12[11] other Oracle middleware products of infringement, Thought's infringement theories for those products are based solely on the fact that they are sold with TopLink. (Hosking Decl., ¶ 61; Artuz Decl., Ex. 2 at 161:3-19.) In other words, Thought does not accuse product features or functions that fall outside TopLink's ORM features and functions. Accordingly, because TopLink does not infringe the Asserted Claims, none of the other accused Oracle products infringes.

[10] The Court struck all of Thought's other theories of infringement. (Dkt. 176.)
[11] *See* Artuz Decl., Ex. 7.

D. Oracle Does Not Directly Infringe Because Oracle Does Not Sell the System and Methods of the Asserted Claims

Direct infringement requires proof that Oracle makes, uses, sells, or offers to sell the claimed inventions of the ’197 patent. 35 U.S.C. § 271(a). Thought’s only ***direct*** infringement theory for which it seeks damages is based on Oracle’s ***sale*** of TopLink or products that are bundled with TopLink. (Artuz Decl., Ex. 5 at 42:17-44:10 and 68:24-70:9.) Oracle, however, does not ***sell*** the claimed inventions, which include “systems” (claims 1, 3, and 5) and “methods” (claims 7 and 8).

First, the sale of a product cannot constitute direct infringement of a method claim. *Mirror Worlds, LLC v. Apple Inc.*, 692 F.3d 1351, 1361 (Fed. Cir. 2012). Rather, Thought must prove actual practice of the method. Oracle’s sale of a product cannot infringe claims 7 and 8.

Similarly, Oracle’s sale of TopLink, by itself, cannot constitute infringement of system claims 1, 3, and 5. TopLink, as distributed to customers, is simply a set of software libraries that may never be used for ORM. A customer might never activate TopLink’s ORM features or connect those libraries to an object application and a database. (Leung Decl., ¶¶ 27-28.) Claims 1, 3, and 5 require actual system components active only when an ORM system is running.[12] (Hosking Decl., ¶ 59.) For example, the following claim elements exist, if at all, only at runtime:

> a first [interface/adapter] ***responsive to*** the object application”, “a second [interface/adapter] … ***in communication with*** at least one data store”, “an application bridge ***receiving*** an object”, “said first [interface/adapter] ***extracting*** the object attributes and the object name as data”, “said second interface ***having*** a meta data map comprising at least one object name”, etc.

(Artuz Decl., Ex. 1 at claims 1 and 3 (emphasis added); Hosking Decl., ¶ 59.) It is not enough that TopLink may, under Thought’s infringement theories, be capable of performing these functions because the Asserted Claims do not include language that requires mere capability. *See ACCO Brands, Inc. v. ABA Locks Mfrs. Co.*, 501 F.3d 1307, 1313 (Fed. Cir. 2007) (no direct infringement where plaintiff failed to show actual use of the accused product in the infringing mode); *Nazomi Commc'ns, Inc. v. Nokia Corp.*, 739 F.3d 1339 (Fed. Cir. 2014) (no infringement of claims that

[12] While infringement of the Asserted Claims is dependent only on the function of the middleware, the claims require that the middleware is actually set up to perform the recited operations.

required a combination of hardware and software where defendant only sold hardware products).

Because Oracle's sale of the accused products cannot constitute direct patent infringement under 35 U.S.C. § 271(a), the Court should grant partial summary judgment of no direct infringement on that issue.

IV. TLS2 INVALIDATES THE ASSERTED CLAIMS

To find the Asserted Claims invalid, the Court need only determine whether the TLS2 prior art disclosed a single feature: "runtime interchangeable" adapters. Thought did not dispute that TLS2 included all other elements of the Asserted Claims.[13] As shown below, TLS2 disclosed adapters that are "runtime interchangeable," and unlike later Java versions of TopLink, TLS2 included the interfaces and adapters of the Asserted Claims.

A. TLS2 Is Prior Art Under Both § § 102(a) and 102(g)

A patent claim is anticipated under 35 U.S.C. § 102(a) if it was "known or used by others in this country" in a manner "accessible to the public[.]" *Minnesota Min. & Mfg. Co. v. Chemque, Inc.*, 303 F.3d 1294, 1301 (Fed. Cir. 2002). A patent claim is anticipated under 35 U.S.C. § 102(g) if it was "invented by another" in this country and was not "abandoned, suppressed, or concealed." *Fox Grp., Inc. v. Cree, Inc.*, 700 F.3d 1300, 1304 (Fed. Cir. 2012). An invention of foreign origin is considered invented in this country on the date it is "embodied in tangible form in the United States." *Scott v. Koyama*, 281 F.3d 1243, 1247 (Fed. Cir. 2002).

Development of TLS2 began in 1995 in Canada. (Leung Decl., ¶ 5.) By 1996, The Object People added an office in Raleigh, North Carolina. (*Id.* at ¶ 6.) In that same year, TLS2 was demonstrated at an industry conference in New York, advertised in this country, sold to American companies, and used in the United States. (*Id.* at ¶¶ 6-12, 15, and 17-20; Hosking Decl., ¶¶ 63-64.) TLS2 was not abandoned, suppressed, or concealed—it was actively commercialized and promoted.

[13] *See* Artuz Decl., Ex. 8, pp. 165-167 and Ex. 9, pp. 27-29. The Court incorporated the concept of "adapters" (and thus the "runtime interchangeable" element) into the constructions for "interface" and "adapter abstraction layer." (Dkt. 116 at 10, 13.) Per its interrogatory responses, Thought's basis for alleging that TLS2 does not have the claimed "interfaces" and "adapter abstraction layer" is based solely on TLS2 allegedly not having adapters that are "runtime interchangeable." *See also* Hosking Decl., ¶¶ 65-73, stating the same.

(*See id.*) Accordingly, TLS2 is prior art to the ’197 patent under both 35 U.S.C. §§ 102(a) and 102(g).

B. TLS2 Anticipates the Asserted Claims

1. TLS2 Disclosed and Used “Runtime Interchangeable” Adapters

The Court’s claim construction requires “adapters” to be “runtime interchangeable.” According to the ’197 patent, a benefit of this feature may be that a user can switch or restructure databases without having to change and recompile object application source code.[14]

Consistent with this explanation, Thought’s expert opined in his infringement analysis that the TopLink components alleged to be the claimed adapters are “runtime interchangeable” because the components are configurable at runtime. (*See, e.g.*, Artuz Decl., Ex. 14 at p. 53 and Ex. 19 at p. 41.) But he also stated that “runtime interchangeability” could refer to actually switching one adapter for another (although he never identifies any instance where this is done in the accused versions of TopLink). (Artuz Decl., Ex. 2 at 89:7-10, 90:1-3.) Finally, Thought’s expert also stated that “runtime interchangeability” is achieved if the mapping metadata is stored outside of the object application’s source code, so the mappings can be modified without recompilation of the object application. (*See, e.g.*, *id.* at 205:14-21.) Under each of these definitions, TLS2 disclosed adapters that are “runtime interchangeable.”

The “runtime interchangeable” adapters in TLS2 included its “Session” and “Accessor” components, which managed the communications between object applications and relational databases. (Hosking Decl., ¶ 76.) When an object application used a Session from TLS2, the interaction between the object application and TLS2 was mediated by the Session and the interaction between TLS2 and a relational database was mediated by an Accessor corresponding to the Session. (*Id.*) When an object application sought to connect with a particular database, it asked TLS2 for a Session. (*Id.*) TLS2 would then create a new Session and return it to the object

[14] According to the ’197 patent, in prior art ORM systems, the object application communicated with a database where the particularities of the database are hard-coded into the object application’s source code. (Artuz Decl., Ex. 1 at 1:42-63.) An attempt to add a new database or modify an existing database may require changing that code, which in turn requires recompilation. (*Id.* at 4:23-33.) The ’197 patent purports to avoid this by insulating the object application from the particulars of a database and by allowing the middleware to handle integrating a new or changed database. (*Id.*)

application. (*Id.*) At the same time, TLS2 would create a new Accessor corresponding to the Session to interact with the database. (*Id.*) When the object application used the Session, the Accessor was effectively invisible to the object application, as the Session insulated the object application from the Accessor. (*Id.*) By using Descriptors, which contained object-to-relational mappings, TLS2 converted object-oriented data into relational data and vice-versa when the object application used the Session to, for example, read from a relational database. (*Id.*)

As shown in more detail below, TLS2 disclosed all three types of "runtime interchangeable" adapters identified by Thought's expert (although disclosure of just any one type is sufficient to invalidate). First, TLS2 allows the configuration details of a particular database to be provided to TLS2 at runtime. TLS2's "login method" provided this capability, and did not require recompilation of the object application.

Second, TLS2 provided interchangeability by switching between different Sessions and Accessors at runtime, and even empowered object applications to simultaneously interact with multiple relational databases. This "runtime interchangeability" allowed an object application to obtain multiple Sessions (each with a separate Accessor) from TLS2 through the use of a common "factory method," and again did not require recompilation of the object application.

Third, TLS2 allowed mappings to be stored outside of the object application's source code. It did this by providing a method of registering Descriptors (containing the mappings) for a particular relational database at runtime, without requiring recompilation of the object application.

(a) The functionality of the first adapter and the second adapter in TLS2 was configured at runtime

Thought's expert first opines that adapters are "runtime interchangeable" where they can be configured based on values that are provided as "parameters" at runtime, and asserts that the inclusion of this feature in the Java versions of TopLink is evidence of infringement. (*See, e.g.*, Artuz Decl., Ex. 14 at p. 53 and Ex. 19 at p. 41.) TLS2 also used methods where values are input as "parameters" at runtime to configure TLS2's Sessions and Accessors. As shown below, TLS2 could change to a different database at runtime, and neither TLS2 nor the object application required recompilation.

Object applications used TLS2 by invoking TLS2's methods. (Hosking Decl., ¶ 77.) Some of TLS2's methods accepted "parameters" as input which came from object applications. (*Id.*) For example, an object application written to use TLS2 invoked TLS2's "login method," which accepted parameters, including a loginToServerName, databaseName, dataSourceName, username, and password. (*Id.*) Those parameters provided to TLS2 at runtime determined which relational database to access. (*Id.*) The login method was not written to require a prescribed loginToServerName, databaseName, dataSourceName, username, or password; rather, it accepted whatever values were provided to the method by the object application that invoked it. (*Id.*)

An application invoked a TLS2 method in at least two ways. (*Id.* at ¶ 78.) First, the values for parameters of a method – for example, the values for a particular loginToServerName, databaseName, dataSourceName, username, or password – could be "hard-coded" into the application's source code. (*Id.*) Among the drawbacks to hard-coding values in this manner was the need to change the object application's source code and recompile the object application if the values provided to TLS2's login method for the parameters needed to be changed. (*Id.*)

The second, more preferable approach, was to use "variables" as the values for the parameters in the object application's source code. (*Id.* at ¶ 79.) A variable is a placeholder in the source code of the object application. (*Id.*) Object applications used variables so that values could be determined at runtime or provided to the object application at runtime, such as by input from a user or from the contents of an external file or database. (*Id.*; Leung Decl., ¶¶ 15 and 18-19.) For example, object applications were written to use variables as the values for the parameters in TLS2's login method. (*Id.*) In these object applications, variables were used in the object application's source code, and their values were obtained at runtime and provided to TLS2's login method as values for the loginToServerName, databaseName, dataSourceName, username, and password parameters. (*Id.*) These object applications prompted a user to enter information such as a server name, database name, username, and password. (*Id.*) Once the user provided this information, the object applications used these values, input by the user at runtime, as the values for the parameters in TLS2's login method. (*Id.*) Using this approach allowed different usernames and passwords to be used without requiring recompilation of the object application's source code. (*Id.*)

The use of variables in an object application and the use of parameters in methods was not novel in March 1997.[15] (Hosking Decl., ¶ 80.) Because the value of a variable was obtained at runtime (rather than a constant value being hard-coded in the object application's source code), and because the method accepted many possible values for a parameter, neither the method nor the object application required recompilation if a user provided different values at runtime. (*Id.*)

Application developers using TLS2 harnessed this flexibility by creating object applications that used variables as parameters. (Leung Decl., ¶¶ 15 and 18-19; Hosking Decl., ¶¶ 79-82.) One customer's object application before March 1997 used TLS2's login method to switch between: (1) a test database to test for software bugs, and (2) a production database to be rolled-out for real-world use. (Leung Decl., ¶ 18.) The user switched between using the test database and the production database without recompiling the object application. (*Id.*)

Another TLS2 customer's object application before March 1997 had a graphical user interface similar to that shown in Figure 5 below in which a user of the object application could provide input at runtime to use different relational databases without recompiling the object application. (*Id.* at ¶ 19.) The object application used the input provided by the user at runtime to form the values for the parameters for TLS2's login method, and connect to the relational database entered by the user. (*Id.*) In this way, a user of this object application provided input at runtime to choose which relational database TLS2 would connect with, and the object application did not require recompilation to be used with different relational databases. (*Id.*)

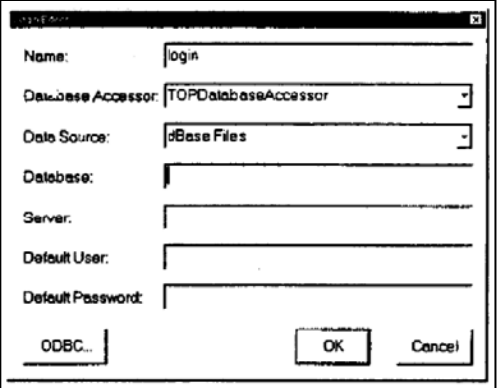
Name:	login
Database Accessor:	TOPDatabaseAccessor
Data Source:	dBase Files
Database:	
Server:	
Default User:	
Default Password:	

Figure 5 — Connect to your database

TLS2 shipped with an object application – the "Descriptor Builder"[16] – that did precisely that: it was designed to switch between databases at runtime. (Hosking Decl., ¶¶ 81-83; Artuz Decl., Ex. 10 at ORCL_THT_01111123; Leung Decl., ¶¶ 12 and 15.) It implemented an interface for users to take advantage of

[15] Dr. Jagadish acknowledges it was "trivial" to modify an application that used hard-coded constant values to instead use variables whose values are set at runtime. (Artuz Decl., Ex. 2, at 119:12-18.)

[16] The Descriptor Builder object application should not be confused with the Descriptors component, which stored the metadata mappings and was part of the middleware itself.

this feature, which was highlighted in TLS2's user manual (as pictured above) and in the source code for the Descriptor Builder application. (*Id.*)

The Descriptor Builder prompted the user for the needed values, such as the type of database and its location. (Hosking Decl., ¶ 84.) After the user input those values, the Descriptor Builder stored them in variables, and used those variables when connecting to the database. (*Id.*) If the user wished to connect to a different database, he provided different values corresponding to the different database. (*Id.*) The Descriptor Builder did not need to be recompiled for the user's input to take effect, which demonstrates the "runtime interchangeable" nature of TLS2's adapters. (*Id.*)

Thus, TLS2 did not require the log-in values for its login method to be hard-coded into every object application's source code that was designed to use TLS2. (*Id.* at ¶ 85.) To the contrary, as discussed above, object applications using TLS2 existed by March 1997 that were designed to receive user input at runtime that was provided as values for the parameters of TLS2's login method to access different relational databases without requiring recompilation of the object applications. (*Id.*; Leung Decl., ¶¶ 15 and 18-19.) Object applications using TLS2 also retrieved data from sources outside of the object application, which was provided to TLS2 at runtime for accessing a particular relational database. (Hosking Decl., ¶¶ 77, 79-86; Leung Decl., ¶¶ 12, 15 and 18-20.) The data would be retrieved from an external file, a database, or a command line argument apart from the object applications themselves, and therefore did not require recompilation of the object applications if the data needed to be changed. (*Id.*) (Hosking Decl., ¶¶ 77, 79-86.)

TLS2 thus did not require that values be hard-coded into the source code of every object application that used TLS2, but instead values could be provided from outside the object application at runtime. (*Id*. at ¶ 86.)

(b) TLS2 allowed applications to interchange between different first and second adapters at runtime

TLS2 also met Thought's second definition of "runtime interchangeable," i.e., *interchanging* (or switching) between adapters at runtime. TLS2 provided multiple Sessions to an object application, which allowed an object application to obtain an initial Session to connect with one database (such as an Oracle database), and to switch that Session with another to connect with a different database (such as a different Oracle database or a Sybase database). (Hosking Decl., ¶ 87.) Each time, TLS2 automatically created a corresponding Accessor to implement access to the respective database. (*Id.*) TLS2 thus provided the use of multiple adapters to access different databases without requiring recompilation of the object application.[17] (*Id.*)

One way TLS2 enabled object applications to use this feature was by providing a "factory method," which used a parameter, to the application. (*Id.* at ¶ 88.) The TLS2 documentation directed a developer to use the factory method for the object application to communicate with a Session. (*Id.*; *see, e.g.*, Artuz Decl., Ex. 10 at ORCL_THT_01111202 and ORCL_THT_01110991; Ex. 27 at ORCL_THT_02343724 ll. 1359-1362 and ORCL_THT_02343740 ll. 2196-2199.) The Session in turn communicated with a corresponding Accessor. (*Id.*) The object application, once written to use TLS2's factory method, was able to use a Session without need for recompilation. (*Id.*) Thought admits that "runtime interchangeability is achievable by use of a factory method pattern." (*See, e.g.*, Artuz Decl., Ex. 13 at ¶ 29.)

Accordingly, TLS2 provided "runtime interchangeable" adapters by allowing the object application to actually switch one adapter for another via the use of the "factory method."

(c) TLS2's Descriptors were "runtime interchangeable"

Finally, TLS2 met Thought's third definition of "runtime interchangeable" adapters by providing a method to register Descriptors with TLS2 at runtime and by not requiring that the Descriptors be hard-coded in the object application's source code, but instead be stored outside of the object application. (Hosking Decl., ¶ 89.) The Descriptors component of TLS2 stored the mapping metadata for translating communications between an object application and a database. (*Id.*) If a database's structure underwent a significant change, the mapping metadata would need to be updated to account for that change. (*Id.*) The manual for TLS2 showed that it was possible to store the mapping metadata in the source code of the object application. (*Id.*; *see, e.g.*, Artuz Decl., Ex. 10 at ORCL_THT_01111268.) In this particular circumstance, editing the mapping metadata could require editing the source code of the object application, which in turn could require

[17] Likewise, the database to which an application connected may be determined at runtime based on information retrieved from a file, a database, or a command line argument. *See, supra*, IV.B.1.a.

recompilation. (*Id.*)

Nevertheless, the undisputed evidence is that the mapping metadata did not need to be stored ***inside*** the object application. (*Id.* at ¶ 90; Leung Decl., ¶ 20.) To the contrary, TLS2 enabled storage of mapping metadata ***outside*** of the object application, and customers stored mapping metadata in this way prior to March 1997 (*Id.*) This included storing mapping metadata in a separate file or database that was accessed at runtime and provided to TLS2's Descriptor registration method at runtime. (*Id.*) In both circumstances, the mapping metadata was runtime interchangeable, because it could be edited without requiring recompilation of the object application. (*Id.*)

TLS2's method for registering a Descriptor at runtime with parameters achieved this capability. (Hosking Decl., ¶ 91.) TLS2 did not prescribe how the value of the Descriptor's parameter was set by the object application. (*Id.*) An object application would invoke the method for using a Descriptor variable, and the value of that variable could be set in numerous ways, such as from the contents of a database or from a file. (*Id.*) The contents of that file or database could change, and thus the resulting Descriptor (with its mapping metadata) would change, without the need to change and recompile the object application's source code. (*Id.*)

Thought cannot legitimately dispute that it was known to store the Descriptors outside of the object application, let alone maintain its position that TLS2's Descriptor registration method ***requires*** that the Descriptors be stored in the object application's source code. TLS2 therefore implemented this third definition of "runtime interchangeable" adapters.

2. Thought Ignores Evidence of Runtime Interchangeability

Thought's expert – who admitted he is not a Smalltalk language expert – asserts that in TLS2 a "particular Accessor ***must*** be selected during design-time and hard-coded into the object application source code." (Artuz Decl., Ex. 2 at 253:25-254:20 (emphasis added); Ex. 21 at ¶ 40 (citing to ORCL_THT_01110991).) But he relied only on one small portion of a lengthy manual for TLS2 in so opining, rather than analyze all relevant evidence. The use of hard-coded values in object applications was not ***required*** by TLS2**,** and Thought's expert ***never analyzed the relevant source code*** of TLS2 in his expert report to determine whether it enabled runtime interchangeability through the use of parameters. (Hosking Decl., ¶ 92.) Nor did he analyze abundant other evidence of TLS2's functionality. (*Id.*) His ostrich-like approach cannot prevent summary judgment.

As discussed above, an object application could obtain a Session (which in turn caused TLS2 to create an Accessor) by invoking the parameterized method. (*Id.* at ¶ 88.) This method required a parameter, but nothing about that method required the parameter's value be hard-coded into the object application. The object application could instead use a variable as the value for the parameter. (*Id.*) This achieved runtime interchangeable adapters, and nothing about this method required that an Accessor be hard-coded into the object application. (*Id.*)

Moreover, as shown in the screenshot above, a user could select the name of whichever Accessor the user wanted to use at runtime. (*Id.* at ¶ 93.) The Descriptor Builder tool, which was shipped with TLS2, provided an example of how to populate the value of a variable so that the Accessor changed at runtime. (*Id.*) Customers stored mappings outside of the object application, allowing editing at runtime without recompilation. (*Id.*)

Oracle provided Thought with access to the TLS2 source code computer as well as printouts of the TLS2 source code,[18] but Thought's expert failed to provide any analysis of the relevant source code in his rebuttal report. (*Id.* at ¶ 94.) Nor does he say that his analysis of the TLS2 source code supports his conclusion that TLS2 ***required every*** object application to hard-code values of parameters in the application's source code. (*Id.*) In the same vein, the sample code in the '197 patent shows parameter values used to obtain and configure adapters that are hard-coded into the object application; nevertheless, Thought's expert testified that those adapters were still "runtime interchangeable." (Artuz Decl., Ex. 2 at 120:1-7, 121:8-23.) Therefore, a single demonstration that an object application using TLS2 ***may have used hard-coded parameter values*** used to obtain or configure adapters cannot end the inquiry as to whether TLS2's adapters are "runtime interchangeable." (Hosking Decl., ¶ 94.) As Thought's expert admitted, the source code is "at the heart of the matter" when analyzing software, and that "source code" is "necessary to speak definitively" about how software operates. (Artuz Decl., Ex. 2 at 201:10-13. *See also id.* at Ex. 21 at

[18] Oracle produced source code for the TLS2 middleware, Descriptor Builder application, and TopLink Demo applications. *See* Artuz Decl., ¶ 45.

¶ 157.) Indeed, he admitted that "[a]ny dispute as to how an actual software release (or any software) functions is ultimately settled, as a factual matter, by reviewing the source code." (*Id.* at ¶ 168.) His failure to do so for TLS2 is both telling and indicate of the lack of factual support for his assertions.

For the foregoing reasons, Thought's assertion that TLS2 does not satisfy the only disputed claim limitation is factually unsupported, conclusory, and does not create a genuine factual dispute. (Hosking Decl., ¶ 95.) Accordingly, the Court should grant summary judgment of invalidity of the Asserted Claims under §§ 102(a) and 102(g).

C. TLS2 also Renders the Asserted Claims Obvious

The Asserted Claims also are invalid as obvious under 35 U.S.C. § 103 in light of TLS2's undisputed functionality and admissions by Thought's expert that it would be "trivial" to modify a "hard-coded" ORM implementation so that the "parameter values" would "come from input variables." (Artuz Decl., Ex. 2 at 120:1-121:7.) Such a "trivial" modification would preclude the need to recompile the source code and would therefore provide the claimed "runtime interchangeable" adapters.

A claim is obvious if it is no more than "the predictable use of prior art elements according to their established functions." *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 401 (2007). The obviousness analysis, which is ultimately a conclusion of law, requires consideration of: (1) the level of skill in the art; (2) the differences between the prior art and the asserted claims; (3) the scope and content of the prior art; and (4) secondary considerations of non-obviousness. *Perfect Web Techs, Inc. v. InfoUSA, Inc.* 587 F.3d 1324, 1327 (Fed. Cir. 2009). Here, the facts relevant to these factors are undisputed, and thus summary judgment of obviousness is appropriate.

1. The Level of Skill in the Art is Undisputedly High

A person of ordinary skill in the art in the ORM field is attempting to solve software development problems encountered by other software engineers. There is no material dispute that the skill level is high, and for the purpose of this motion Oracle accepts Thought's view that a person of ordinary skill in the art is a software engineer with "significant knowledge in both object-oriented programming and relational database technology." (Artuz Decl., Ex. 13 at ¶¶ 22-23; *see* *also id.* at Ex. 21 at ¶ 28 ("the level of skill in the relevant art is high").)

A "higher [] level of skill" favors a determination of "obviousness." *See Innovention Toys, LLC v. MGA Entm't, Inc.*, 637 F.3d 1314, 1323 (Fed. Cir. 2011). This is because a more sophisticated person of skill in the art is more likely to draw on ideas from divergent sources, "without being told to do so." *DyStar Textilfarben GmbH & Co. Deutschland KG* v. *C.H. Patrick Co.*, 464 F.3d 1356, 1370 (Fed. Cir. 2006). As Thought's expert confirmed, "[T]he higher the level of ordinary skill, the easier it may be to establish that an invention would have been obvious." (Artuz Decl., Ex. 21 at ¶ 9.) Thus, there is no material dispute on this issue.

2. There is Only a Single Alleged Difference Between the Claims of the '197 Patent and TLS2

As discussed above, Thought concedes that TLS2 discloses every element of the Asserted Claims, with the sole exception of whether the adapters were "runtime interchangeable." While Oracle has shown that this element is also present, at most the scope of the the alleged difference between the claims and the prior art is extremely narrow, weighing in favor of obviousness as a matter of law. *See Scanner Techs. Corp. v. ICOS Vision Sys. Corp. N.V.*, 528 F.3d 1365, 1382 (Fed. Cir. 2008) (upholding obviousness finding where the "relatively small logical gap between the prior art and the claim" was "closed by a person of ordinary skill in the art pursuing known options").

3. Using Variables Set at Runtime Instead of a Constant Hard-Coded into an Application Was Well-Known by March 1997

(a) The motivation in the art to de-couple object applications from data sources was firmly established

"When there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp. If this leads to the anticipated success, it is likely the product not of innovation but of ordinary skill and common sense." *Ball Aerosol & Specialty Container, Inc. v. Limited Brands, Inc.*, 555 F.3d 984, 991 (Fed. Cir. 2009) (*quoting KSR*, 550 U.S. at 421.) By March 1997, it had been well-known for years in the ORM field that it was desirable to de-couple an object application from any single database to allow software developers more flexibility in migrating between data sources. (Hosking Decl., ¶ 97.) And there is no dispute that the methods of achieving this decoupling were well-known and in fact "trivial" to accomplish.[19] Based on this knowledge, a person of ordinary skill in the art would have been motivated to modify TLS2 to provide runtime interchangeable adapters, and would have encountered no difficulty achieving that goal. (*Id.*)

As early as 1992, former Apple CEO Steve Jobs, then CEO of NeXT, Inc., said of NeXT's "DBKit" or "Database Kit" ORM solution (which endeavored to solve the exact problem identified by the '197 patent) that it provided "adaptors that click in to the back end of the . . . architecture," so that "[o]nce a developer writes their application to the Database Kit set of objects, you can actually switch between Sybase or Oracle or any other database without changing a line of code in the application." (*See* Artuz Decl., ¶ 46.) DBKit "insulates the application developer from whatever backend the customer may prefer to use." (*Id.*) DBKit thus allowed an object application to be designed in a database-independent fashion. (*Id.*)

Thus, the problem in the art was not only identified long before March 1997, but was already addressed by numerous solutions. Consistent with this established motivation, there is significant evidence that the benefits of avoiding hard-coded middleware were well-known, and could be achieved by storing ORM metadata mappings outside of the object application. The Object People's presentations highlighted the importance of defining interfaces and Descriptors independently from the object application. (Hosking Decl., ¶ 97; *see, e.g.*, Artuz Decl., Ex. 39 at pp. 4, 7, and 12.) Likewise, public on-line postings, magazine articles, and books highlighted the usefulness of storing ORM metadata outside of the object application to enable database independence. (S*ee, e.g.*, Artuz Decl., ¶ 46 and Exs. 28 at ORCL_THT_01311709; 34; 35; 39 at pp. 4, 7, and 12; 40 at pp. 4-7; and 36.) The industry was thus clearly motivated to provide runtime interchangeability before the '197 patent.

(b) Thought admitted that the implementation of runtime interchangeable adapters was "trivial"

Thought has no right to a patent offering no more than "the predictable use of prior art

[19] As discussed at length in the anticipation section, the well-known methods including reading in mappings from a file or database outside of the object applications, as well as the ability to obtain database login parameters for an adapter from a file or a database.

elements according to their established functions." *KSR*, 550 U.S.at 417. The "runtime interchangeable" solution claimed by the '197 patent implements quintessentially "predictable" functionality—even Thought's expert explicitly admitted that it would be "***trivial***" to modify a hard-coded solution to use variables to avoid the need for recompilation of the object application. (Artuz Decl., Ex. 2 at 120:1-121:7.) The inventor of the '197 patent also testified that the modification would be a "minor, trivial step." (*Id.* at Ex. 4 at 115:20-116:2.)

Thought's expert opined that TLS2 lacks "runtime interchangeability" for only one reason—TLS2 allegedly used values that are "specified at design-time and hard-coded into the source code of the object application," meaning that to change databases it would be necessary to "update" and "recompile the source code." (*See, e.g.*, Artuz Decl., Ex. 21 at ¶ 67.) There is no dispute that an adapter avoids this where it is "parameterized," and the application can perform its operation via the use of a "variable" that is provided at runtime, rather than through using a hard-coded constant value. (Artuz Decl., Ex. 2 at 46:14-49:03; 117:19-118:6-18.)

At his deposition, Thought's expert admitted that the sample code attached to the '197 patent as Appendix A, which purports to represent an exemplary implementation of the patent, uses the same "hard-coded" implementation he criticized as not enabling "runtime interchangeable" adapters. (Artuz Decl., Ex. 2 at 116:02-118:04.) He agreed that the sample code in the patent implemented "hard-coded" or "constant values," and "you would need to recompile the application code in the demo" to change the code. (*Id.* at 120:1-14.) When asked whether one would be able to change the hard-coded implementation so that the values could be provided at runtime and avoid the need for recompilation, Thought's expert conceded, "it would be very easy to make these parameter values variables that you could set to whatever you want it to set to." (*Id.* at 118:06-18.) "[H]aving parameter values come from input variables," he continued "would be trivial to do." (*Id.* at 120:24-121:7.) Thus, implementation of "runtime interchangeable adapters" via the use of a parameter was not only a "predictable" solution, but it was "trivial." (Hosking Decl., ¶¶ 97-98.)

Claims that add only "trivial improvements that would have been a matter of common sense to one of ordinary skill in the art" are obvious as a matter of law. *Western Union Co. v. MoneyGram Payment Sys., Inc.*, 626 F.3d 1361, 1372 (Fed. Cir. 2010). Thought concedes that TLS2 is identical to the claimed invention in every respect aside from its supposed lack of "runtime interchangeable" adapters, and the admission that it would be trivial to modify non-"runtime interchangeable" adapters into adapters that practice the claims, combined with the undisputed evidence of motivation to do so, compels a finding of obviousness as a matter of law. (*Id.*)

D. Alleged Secondary Considerations of Non-Obviousness Do Not Create an Issue of Fact

There are no significant secondary considerations that can override Oracle's strong *prima facie* showing of obviousness. *See Iron Grip Barbell Co. v. USA Sports, Inc.*, 392 F.3d 1317, 1325 (Fed. Cir. 2004). Moreover, Thought can point to no evidence showing a nexus between any alleged secondary considerations and the ***claimed*** inventions. (Hosking Decl., ¶ 99.) A nexus is required for such a secondary consideration to be significant. *Tokai Corp. v. Easton Enters., Inc.*, 632 F.3d 1358, 1370 (Fed. Cir. 2011). No embodiment of the '197 patent has ever experienced significant commercial success. Sales of Thought's product, CocoBase, were never significant, and began "drying up" by 2005 at the latest. (Artuz Decl., Ex. 3 at 31:03-25.) Thought did not identify any evidence showing that the claimed invention drove sales of any software, nor can it point to evidence providing a nexus between any alleged secondary consideration and the claimed inventions. The lack of significant secondary considerations of non-obviousness means Thought cannot preclude summary judgment on that front.

Thus, because there are no disputes of fact and the conclusion of obviousness is one of law, the Court should grant summary judgment of obviousness.

V. CONCLUSION

For the reasons stated above, summary judgment of non-infringement of the accused products and summary judgment of invalidity is warranted as to each of the Asserted Claims.

DATED: February 25, 2016

Respectfully submitted,

KILPATRICK TOWNSEND & STOCKTON LLP

By: */s/Robert J. Artuz*
ROBERT J. ARTUZ

Attorneys for Defendants
ORACLE CORPORATION, ORACLE AMERICA, INC.
and ORACLE INTERNATIONAL CORPORATION